→ more important note and Week Lecture / homework / recording / assignment

→ Data Structure

→ Heading

Every week from new page

→ Subheading or important point

→ links

Ujjwal Maheshwari

github.com / Ujjwal2327 / DSA-SUPREME

CODES

linkedin.com.in / ujjwal-maheshwari-164886202

CONNECT

Bhaiya Bday
20 June 1998
7:15 AM

https://www.linkedin.com/in/ujjwal-maheshwari-164886202/

https://github.com/Ujjwal2327/DSA-SUPREME

If you like these notes,

Give job opportunities to me

And give party to bhaiya after placement

# IMPORTANT POINTS

- Focus on work on skill and build networks those work in companies and can talk to HR for you
- Internship- do if you want to learn or convert it as PPO
- Rough sols. and dry run are very important
- Do documentation to promote and mail everything you dismss with HR or team
- Think twice, Code once
- To clarify any code you are confused, use cout statements every where to know what is going on in the code
- Code all approaches you can think of and can find & understand from google
- Revise all incorrect & skipped questions in quizes regularly
- Watch sol. only after attempting the question
- Interviewer will ask Time & Space complexity after every sol. you give

→ 2 websites
  → cppreference
  → cplusplus.com

→ Think on paper → with 5 testcases atleast

→ Write readable codes

→ In interviews, tell approaches of questions using example

→ Signs of beginner → no logic build
  cant solve new ques
  forget the approach
  memorizing the ans

↓

Improve → with more no. of questions
  with time
  with dry run on mind logic, not on code
  with your new approach (even brute force)
  with a lot of practice on syntax
  with consistency

→ Focus on placement, not on feeling

You are not studing to get fun

You are studing to get placement

First placement, then fun and interest.

Also focus on health of you and your parents

Bhaad me gya interest yaar, placement ke baad krna

Jo duniya sunna chahati hai, use sunado

Growth is important

Week 4 [Connect] Class 1:45 - 2:01

→ 20-30 interview experience
└→ to break Google pattern

→ Web Dev and DSA ⟶ both are important

CP ⟶ do if you enjoy it.

→ At least learn one more approach if you are doing questions with map

Many times if you tell map approach, interviewer asks another approach

- Tag all khatarnak questions
  - Otherwise you will forget them after some time
- Revise and code all, every after 2 weeks
- For 6th sem student (like me),
  - At least do potd on leetcode everyday along with course
- Make notes
- In dev, you should have at least 2 major projects
  - 1 → in dot batch
  - 2 → group project if you want
- Focus on Networking,
  After 5 months, you should have atleast 2 friends in every company
- All questions doing in batch, should be on your tips

→ Focus on accuracy more than speed

→ Resume is made company specific

→ Make sure other dont take your credits

→ Higher position in career $\propto \frac{\text{Development}}{\text{DSA Complexity}}$

→ Interview experience before interview

→ Flex se kuch nhi hota

Comparision se kuch nhi hota

# IMPORTANT C++ NOTES

- Making global variable is BAD PRACTICE
- To increase range of int / long long, you can use unsigned int / long long
- % → heavy operator
  - → so try to use it less

  Use bitwise operators instead of this if possible
- arr[n] ⟶ BAD PRACTICE

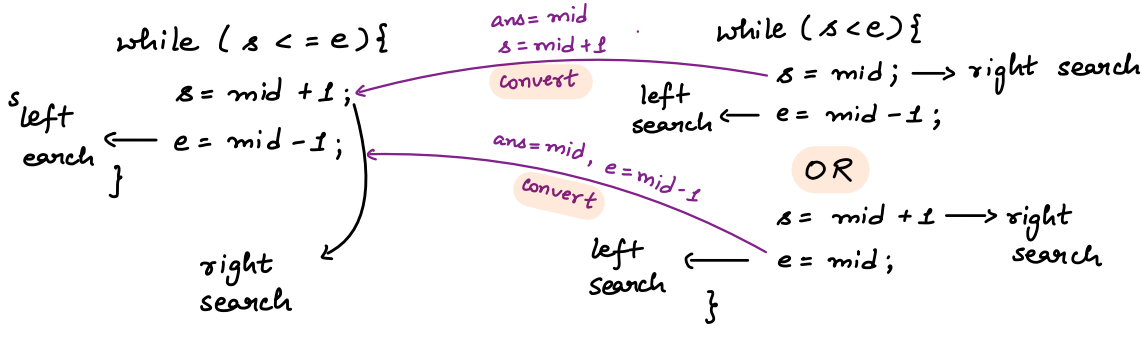
  arr[100000] is better than arr[n]
- to find min., start ans from INT_MAX
- to find max, start ans from INT_MIN
- n & 1 gives rightmost bit of n
- xor → cancels out same elements

  0 ^ ans = 0
  - → 0^1 = 1
  - → 0^0 = 0
- In ASCII ⟶ '0' ⟶ 48
  - 'A' ⟶ 65
  - 'a' ⟶ 97

→ Search Space

- → find range of search space (start & end) in ques of. Binary Search Questions
- → store mid in ans if needed

→ In binary search questions




→ Types of ques in binary search

- → 1st type → classic questions
- → 2nd type → find in search space (range)
  - → predicate function
    - logic to decide either left or right
- → 3rd type → observation in index

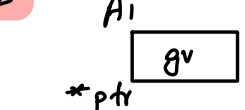
→ In sorted array, try to apply binary search or 2 pointer / 3pointer approach

→ To maximize or minimize
try to use binary search using concept of search space

- → find search space of answer
- → find mid
- → if isPossibleAns(mid) → go to left / right acc. to the ques.

→ int * ptr; → VERY BAD PRACTICE → Segmentation fault

cout << ptr; → gv



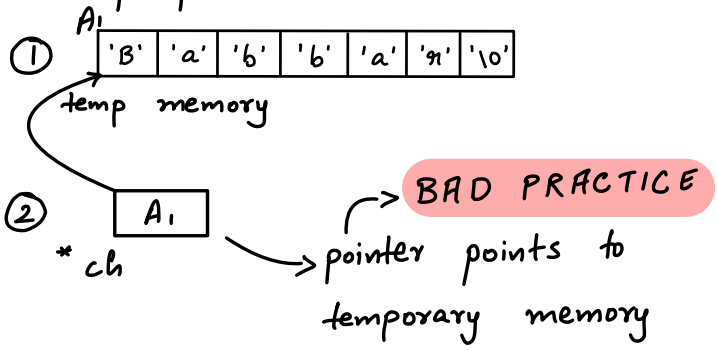

ptr points to a memory location that may not be of its program

Segmentation fault -
- → using other's memory

Use NULL pointer in these situations to initialize

→ char * ch = "Babbar"

↳ 2 step process




→ pass by value but return by reference

int & ref ( int temp ) {

return temp ;

}

BAD PRACTICE → ERROR

returning a variable stored in temporary memory

→

```
int main () {
    int * ptr = solve ();
    return 0;
}
```

A2 | A1 → ptr

pointing to a temp memory, 'a' variable will finished outside solve function

↳ BAD PRACTICE

```
int * solve () {
    int a = 5;
    int * p = &a;
    return p;
}
```

A2 | 5 → a

A3 | A1 → p

## Magical line for Recursion

Ek case solve krdo, baaki recursion sambhal lega

- Just believe on it, dont doubt on recursion
  - Dont go depth inside recursion tree

→ func ( vector <int > arr, int i, int & ans)

pass by reference

if we want to retain value of ans after function call

→ Try to pass everything pass by reference IF POSSIBE

- SC ↓es
- TC ↓es (no copying of variables again & again)

→ to reverse answer, you can use recursion

LET'S GO

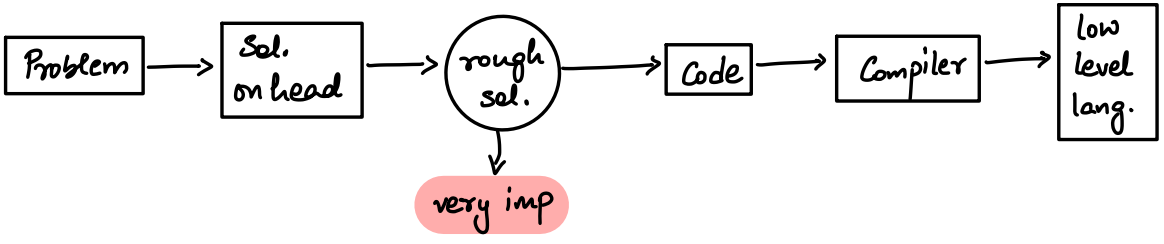
# W1-L1

**Thought process to solve a problem-**

→ Understand a problem

→ input values

→ find approach

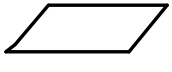


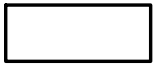

**Algorithm-** Sequence of steps

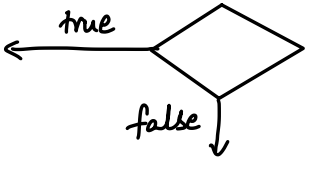
**Flowchart-** Graphical representation of algo

**Components-**

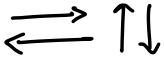

terminator for start /end

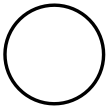

for input /output read /write


computation / process / declaration



decision making block takes condition


flow



Connector

takes function

**Pseudo Code-** Generic way of writing algo

Dry Run → Very important to understand any topic

# W1-L2

**IDE-** Replit, VS-Code

```
# include <iostream>
using namespace std;
int main (){
    cout << "Namaste Bharat";
}
```

preheader file contains implementation of identifiers

using standard namespace implementation of cout choosing from multiple types of namespace

region where scope of identifier is defined

to end any statement

string

used to print on console/ standard display

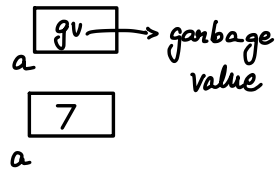
cout << endl; → for next line

cout << '\n';

int a ; ⟶ a is an integer

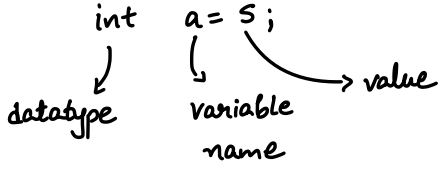
cin >> a ; ⟶ input a from user

ex-7



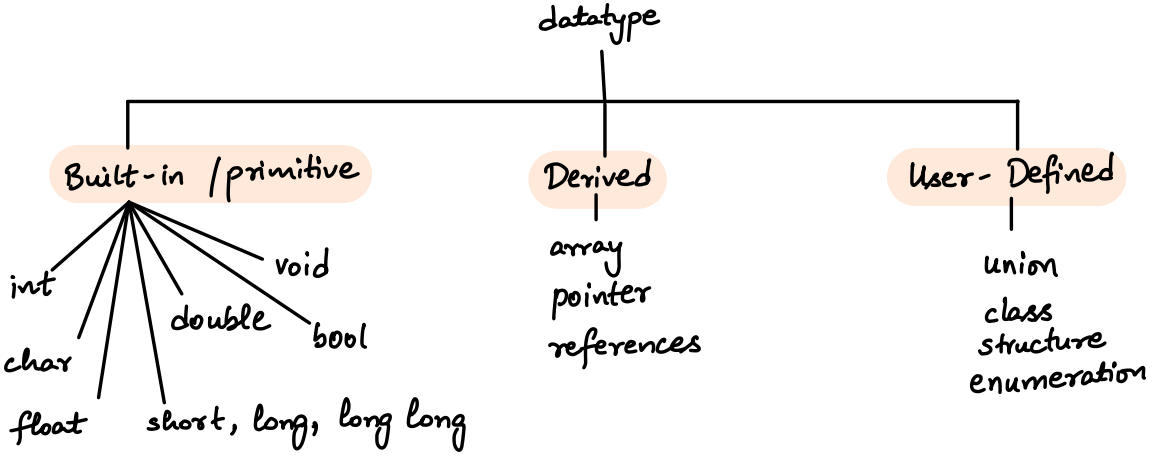

## Variables

named memory location




## Datatypes

type of data




**int** - 4 byte - 32 bits in memory

- → $-2^{31}$ to $2^{31}-1$ in signed int
- → 0 to $2^{32}-1$ in unsigned int

**char** - 1 byte - 8 bits in memory

- → $2^8$ different chars.

ASCII
↳ char maps with numerical ASCII value

char ⟷ ASCII value ⟶ store in memory

bool → 1 byte → 8 bits

true - 1
false - 0

↳ because minimum addressable memory is 1 byte
We cannot address 1 bit in memory

float → 4 byte → 32 bits

double → 8 byte → 64 bits

long long → 8 byte → 64 bits

short → 2 byte → 16 bits

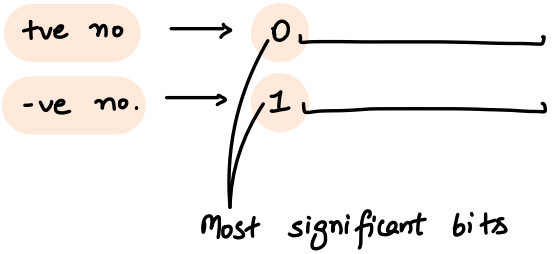
long → 4 byte → 32 bits

## How data is stored

```
int a = 5;
```

↳ 32 bits  0...00101

29 bits




## How -ve number is stored in memory

In 2's complement form

→ 1's complement + 1

→ reverse all bits

```
int a= -7;
```

7 ⟶ 0.....00111 } 32 bits

ignore -ve sign

find binary equivalant

1's (7) ⟶ 1.....11000

find 2's complement

2's (7) ⟶ 1.....11001

→ this is how -7 will be stored in memory

## How to read -ve no. present in memory

→ take 2's complement

1....11001

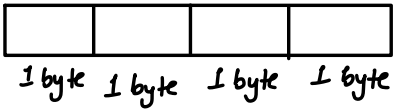
→ 1's complement ⟶ 0...00110

2's complement ⟶ 0...00111

→ +7

-7

## Interesting problem




how computer know these are 4 chars or a single integer

- Using datatype
  - tell 2 things
    - type of data used
    - space used in memory

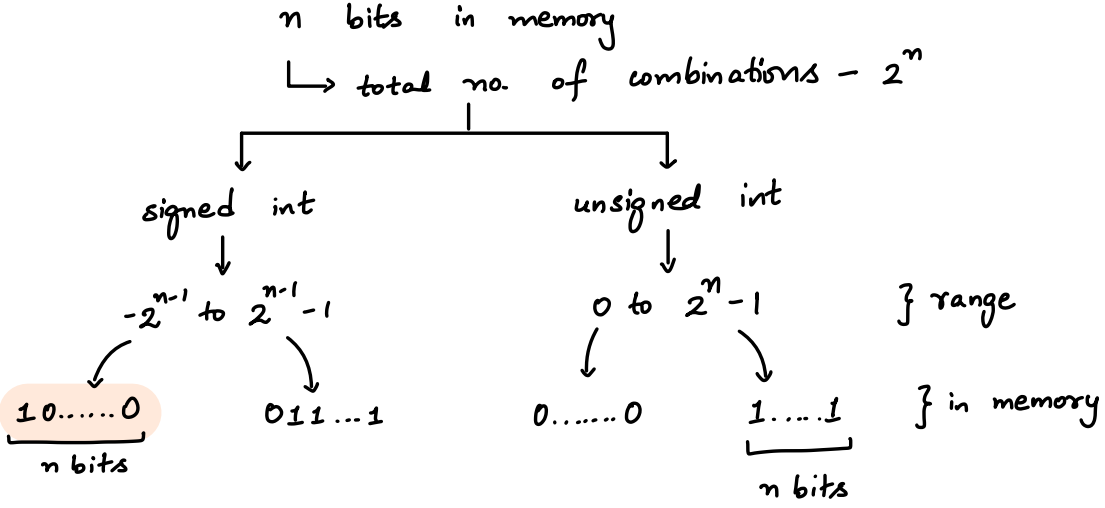
## Signed vs Unsigned

- Unsigned → 0, +ve
- Signed → +ve, -ve, 0
  - by default

int - 4 byte - 32 bits in memory

- total no. of combinations - $2^{32}$
  - signed int
    - $-2^{31}$ to $2^{31}-1$ } range
      - 1 0...0
      - 0 1 1 ... 1 } in memory
      - 2's → 1 0...0 → $-2^{31}$
  - unsigned int
    - 0 to $2^{32}-1$ } range
      - 0.....0
      - 1....1 } in memory

## General Formula




## Typecasting

↳ convert one type of data to another

### implicit typecasting

```
ex- char ch= 97;
    cout << ch;
```
⟶ (a)

### explicit typecasting

```
ex- char ch = (char) 97;
    cout << ch;
```
⟶ (a)

### overflow ex-

```
char ch = 9999;
cout<< ch;
```

9999 ⟶ (binary conversion) 100 111 0000 1111

stores only last 8 bits

so ch stores 00001111 in memory

↓

7

↓

acc. to ASCII table

## Operators -

### Arithmatic Operator

↳ +, -, *, /, %,

int op int ⟶ int

float op int
int op float
float op float
} ⟶ float

double op int
int op double
double op double
float op double
double op float
} ⟶ double

bool op bool ⟶ int

bool act as 0 or 1

3 → int
3.0 → float / double ⟶ by default ⟶ cout << sizeof(3.0);

not int

↓

8

### Relational Operator

a op b

>, <, >=, <=, !=, ==

these are different things

Output – 0 or 1

false ↙ ↳ true

## Assignment Operators

=

## Logical Operators

↳ when you have multiple conditions

a && b ⟶ and ⟶ true if both are true

a || b ⟶ or ⟶ true if any one is true

! a ⟶ not ⟶ negate the result

Output - 0 or 1

false ↙ ↘ true

(cond 1 && cond 2 && cond 3)

if cond 1 is false

compiler will not check further as ans will already false

(cond 1 || cond 2 || cond 3)

if cond 1 is true

compiler will not check further as ans will already true

## Conditions

W1-L3

```
if (cond.){
    execute
}
```

if

```
if( cond ){
    execute 1
}
else {
    execute 2
}
```

if - else

```
if (cond 1)
    execute 1
else if( cond 2)
    execute 2
```

if - else if

```
if (cond 1)
    execute 1
else if ( cond 2)
    execute 2
else if (...
else
    execute n
```

if - else if - else

```
if (cond 1)
    execute 1
else {
    if ( ) { }
    else ( ) { }
}
```

nested if - else

## Loops

↳ to do something repeatedly

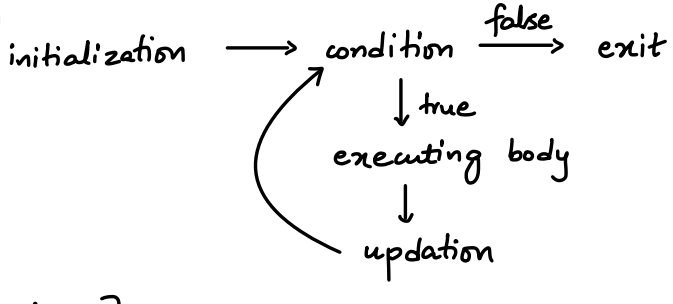
### for - loop

initialization, condition, updation

```
for ( int i=0 ; i<5 ; i=i+1 ) {
    cout << "Love" ;
}
```

→ executing body

### flow -




initialization
condition
updation

} none is mandatory
one or multiple i, c, u can be added

multiple c → i>5, i<10; → i>5 && i<10

**patterns -**

Generally 2 loops ⟶

```
outer loop(){          → for rows
    inner loop(){      → for cols
    }
    cout << endl;
}
```

→ a op= b ⟶ a = a op b

op ⟶ +, -, *, %, /

**cin in if( )**

```
int num;
if(cin >> num){
    cout << "hello";
}
else{
    cout << "hi";
}
```

it will not give error

output -

hello

for all values of num ↓

0, +ve, -ve

**cout in if( )**

```
int num = 0;
if(cout << num << endl){
    cout << "hello";
}
else{
    cout << "hi";
}
```

it will not give error

output -

0

hello

for all values of num ↓

0, +ve, -ve

HLL - High level language

↳ human readable and user friendly

**W1-L4**

C++, C - Middle Level language

namespace ⟶ to avoid collision

↓

multiple definitions of a single keyword

**heirarchy**

↳ various namespaces

↓

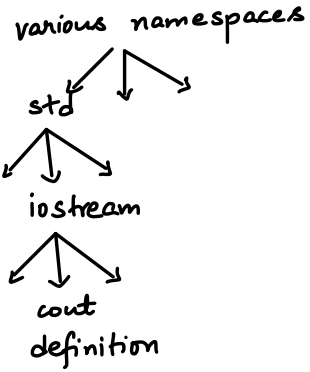
std namespace

↓

iostream preheader file

↓

keyword definition




```
float f = 2.0 + 100;
cout << f;
```

⟶ output -

102 or 102.0

compiler dependent

```
float f = 2.7;
int n = 157;
int diff = n-f;
cout << diff;
```

output -

154

explanation -

n-f = 157 - 2.7 = 154.3

int diff = n-f

diff = 154

# W1-HW

## ternary operator -

↳ syntax

variable = (condition) ? expression2 : expression3

(condition)? variable = expression2 : variable = expression3

2 and 3 cant be statements, they must be exp.

ex- return (a > b) ? a : b ; ⟶ CORRECT

(a > b) ? return a : return b; ⟶ WRONG ERROR

statements, not expression

## by default -

cout << sizeof (2.3) ; ⟶ 8

↳ float

cout << sizeof (a) ; ⟶ 4 ⟶ int

↳ $-(2^{31}-1)$ to $2^{31}-1$

cout << sizeof ($-2^{31}$) ⟶ 8

↳ long long

patterns

# W2-L1

↳ how to think

→ finding formula for rows and cols

n=5

| row | stars |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

→ formula- 0 to < n-1

| n-1 |
|---|
| -1 |
| 0 |
| 1 |
| 2 |
| 3 |

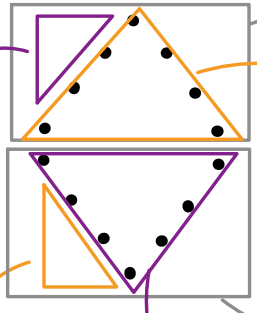
| no. of times loop runs |
|---|
| 0 |
| 0 |
| 1 |
| 2 |
| 3 |

→ as condition fails (0<-1)

→ to do anything n times

↳ for( i=0 ; i<n; i++){}

→ break the complex patterns



upper half

0 to < 2i+1

| chars | |
|---|---|
| 0 → 1 | 1 |
| 1 → 3 | 2(1)+1 |
| 2 → 5 | 2(2)+1 |
| 3 → 7 | 2(3)+1 |

0 to < n-(i+1)

| space | |
|---|---|
| 0 → 3 | 4-1 |
| 1 → 2 | 4-2 |
| 2 → 1 | 4-3 |
| 3 → 0 | 4-4 |

lower half

0 to < i

| space | |
|---|---|
| 0 → 0 | i |
| 1 → 1 | i |
| 2 → 2 | i |
| 3 → 3 | i |

0 to < 2n-2i+1

| chars | | |
|---|---|---|
| 0 → 7 | 8-1 | 8-1 |
| 1 → 5 | 8-3 | 8-(2(1)+1) |
| 2 → 3 | 8-5 | 8-(2(2)+1) |
| 3 → 1 | 8-7 | 8-(2(3)+1) |

# W2-L2

## Bitwise Operators

↳ use on bit level

| | | |
|---|---|---|
| And | (a & b) | 1 if both bits are 1 |
| Or | (a \| b) | 1 if any or both bits are 1 |
| not | (~ a) | negate the result |
| xor | (a ^ b) | same values → 0<br>diff. values → 1 |

~5

↳ 5 → 0...0101

~5 → 1...1010

↳ how compiler read this

↳ 2's complement

0....0101 → 1's complement

0...0110 → 2's complement

-6

So ~5 = -6

## Left and right shift operators

<<

shift all bits to left

* by 2 (not in every case)

↳ if MSB is 1 and 2nd MSB is 0

>>

shift all bits to right

/ by 2 (not in every case)

↳ in -1

a = a << b  a left shifts, b times → result → $a \times 2^b$

a = a >> b  a right shifts, b times → result → $\frac{a}{2^b}$

b cant be -ve

↳ in case of -ve

↳ gives gv

```
a = 5;
a = a << 1;      a = 10

a = 5;
a = a << 2;      a = 20
```

in left shift → filled with 0

in right shift → filled with 0 and 1

in +ve no. ← 0 and 1 → in -ve no.

right shift in -ve number

-ve no. in memory → 1 .....

↓ right shift

11....

↳ signed bit is used to fill the vacant bit

ex-

5 → 0...0 101

-5 → 1...1 011

-5 >>1 → 1...101 → -3

-1 >> 1 → -1

left shift in number where MSB is 1 and 2nd MSB is 0

no. → 10. . → -ve no.

left shift ↳ 0....

↳ +ve no.

## Pre- Post → Increament / Decreament Operator

**pre- increament**

↳ ++a

↳ first increament by 1, then use

**post - increament**

↳ a++

↳ first use then increament by 1

**pre- decreament**

↳ --a

↳ first decreament by 1, then use

**post - decreament**

↳ a--

↳ first use then decreament by 1

```
int a= 5;
cout << (++a) * (++a) ;
```

output -

49

↳ due to operator precendence

↳ Links.txt in repo

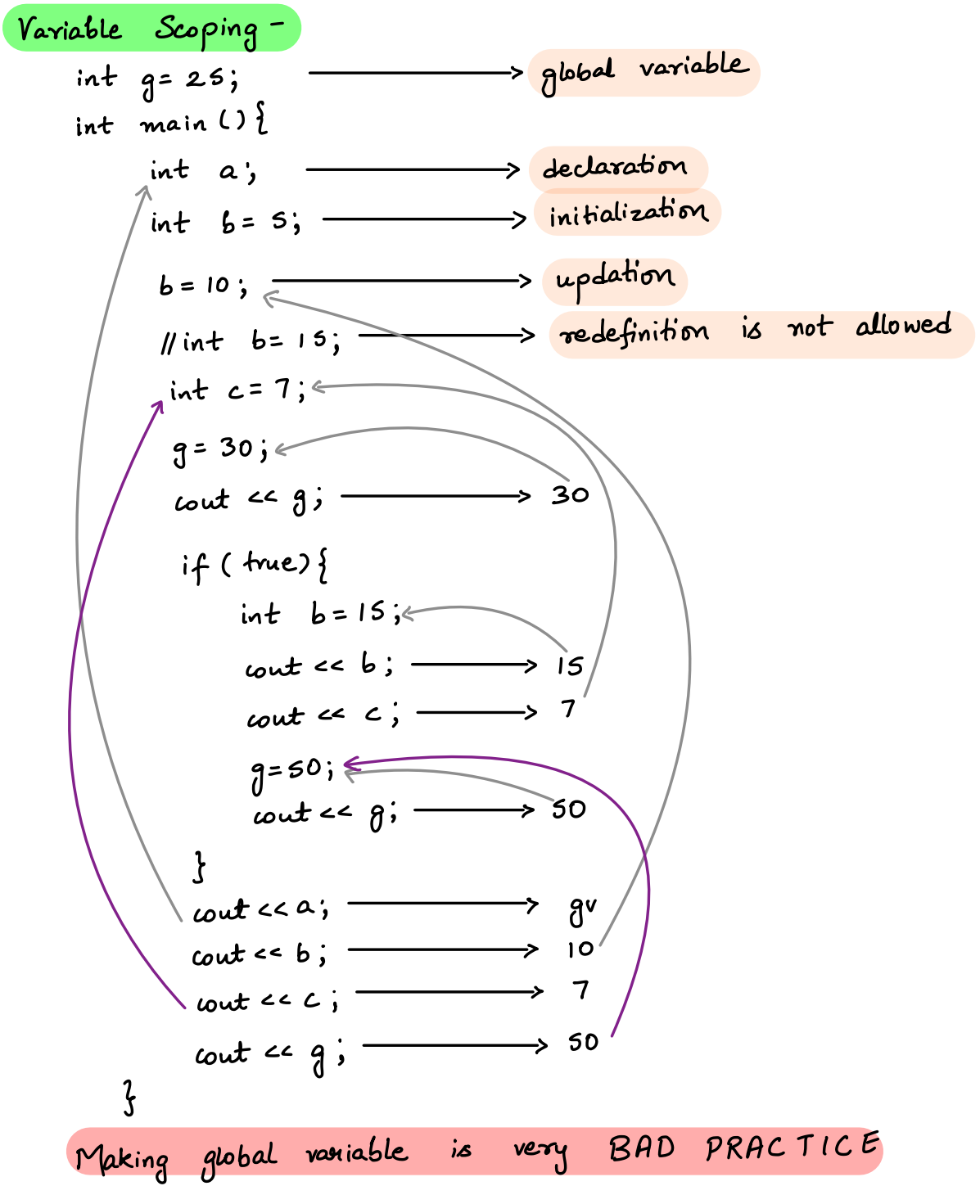
## break and continue

**break**

↳ exit from that loop

**continue**

↳ skip that iteration


Variable Scoping -
int g= 25;
global variable
int main (){
int a;
declaration
int b= 5;
initialization
b = 10;
updation
//int b= 15;
redefinition is not allowed
int c= 7;
g= 30;
cout << g;
30
if ( true){
int b= 15;
cout << b;
15
cout << c;
7
g=50;
cout << g;
50
}
cout << a;
gv
cout << b;
10
cout << c;
7
cout << g;
50
}
Making global variable is very BAD PRACTICE

## Operator Precedence

- order of priority of operator
- no need to remember
  - use brackets properly

## Switch Case

can also have nested switch case

```
switch (expression) {

    case value1 :
        executing body 1
        break;
    case value2 :
        executing body 2
        break;
    :
    case value n :
        executing body n
        break;

    default :
        executing body
}
```

not mandatory

without break
- all below executing body will also execute

→ continue cannot be used in switch case
- can only use in loops

# W2-L3

## Function -

- program linked with well defined task
- why
  - reusable
  - readable
- without
  - bulky
  - lengthy
  - buggy if mistake in any place

syntax -

```
return type functionname ( input parameters) {
        function executing body
}
```

void → empty / no value

```
int main(){
    return 0;
}
```

- return 0; → returns 0 to Operating System
- 0 is used as means of successful execution

→ a cpp file cant have more than 1 main functions

→ main cant have return type other than int in offline compiler

# Function Call Stack

function call ⟷ function invoke

Stack
- Last In First Out

tells
- what functions
- which function calls which
- local variables of function
- return type of function

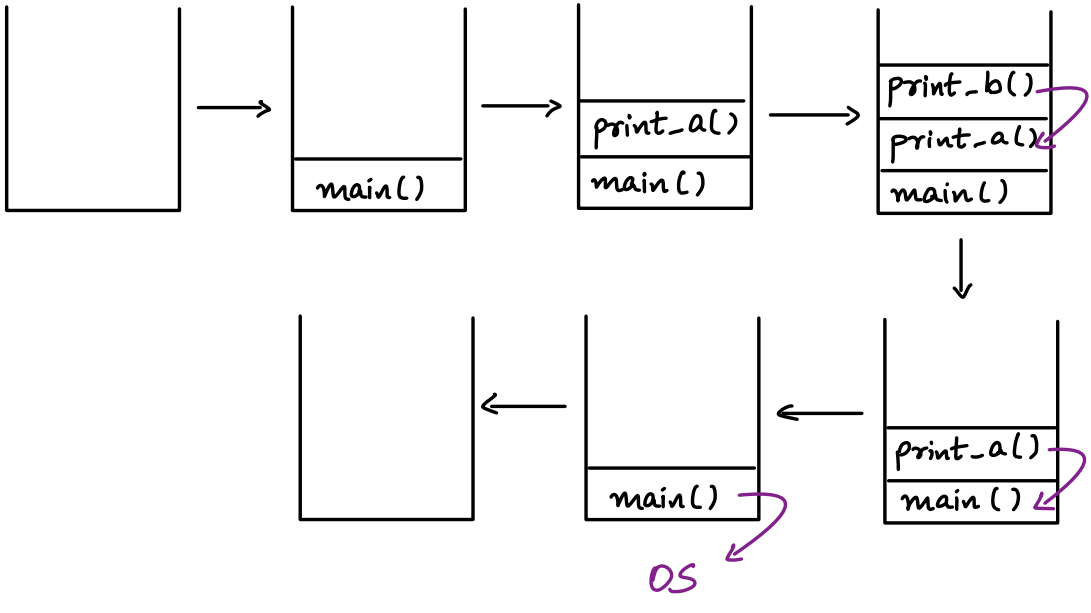
ex-

```
int main(){
    int a=5;
    print_a(a)
    return 0;
}
```

return to OS

```
void print_a(int a){
    cout<< a;
    int b=3;
    print_b(b);
}
```

```
void print_b(int b)
{
    cout<< b;
}
```



## Pass by value

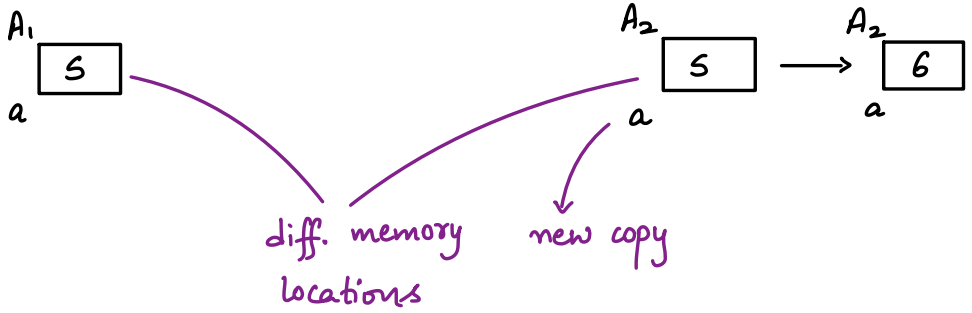
↳ a copy will be created of variables

```
int main() {
    int a = 5;
    printNumber(a);
    cout << a;
}
```

argument

```
void printNumber(int a){
    cout << a;
    a++;
    cout << a;
}
```

parameter




## Address Of Operator &

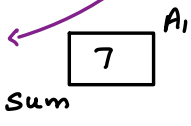
```
int n = 5;
cout << &n;
```

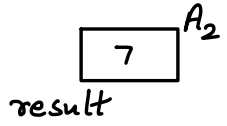
⟶ output - address of n

```
int main(){
    int a = 2;
    int b = 3;
    int sum = add(a, b);
    cout << sum;
    return 0;
}
```

OS

```
int add(int a, int b){
    int result = a + b;
    return result;
}
```



# Function Order

## Order 1

```
int add (int a, int b) {
    return a+b;
}
int main () {
    int a= 3;
    int b = 5;
    int sum= add (a, b);
    cout << sum;
    return 0;
}
```

function declaration and definition

## Order 2

function declaration

```
int add (int a , int b);
int main () {
    int a= 3;
    int b = 5;
    int sum= add (a, b)
    cout << sum;
    return 0;
}
int add (int a, int b) {
    return a+b;
}
```

function definition

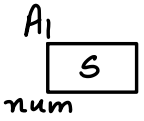
% operator ⟶ heavy operator

↳ so try to use it less

# W2-L4

BTS → Behind The Scenes

```
int num=5;
```




**Symbol table**

int num ⟷ $A_1$

- int → datatype
- num → variable name
- $A_1$ → memory location of variable

variable mapping with address is stored

```
cout <<num;
```

go to → symbol table → go to → $A_1$ (address) → print → 5 (value)

**while loop**

```
int i=0;
while (i<5){
    cout<< i <<endl;
    i++;
}
```

- `int i=0;` → initialization
- `while (i<5){` → condition
- `cout<< i <<endl;` `i++;` → body
- `i++;` → updatation

**flow**

initialization → condition → (false) → exit

condition → (true) → body (containing updatation) → condition

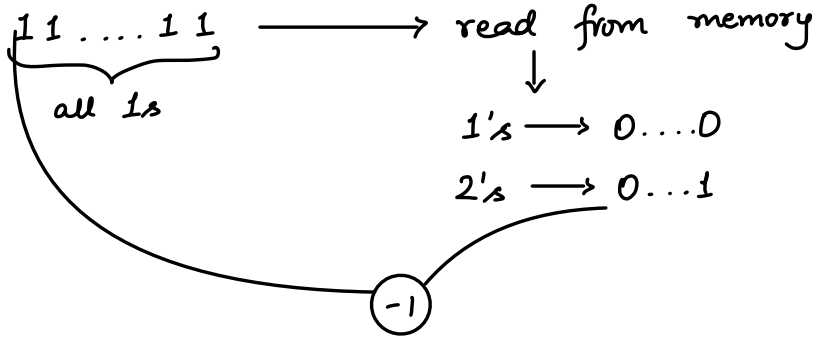
## left and right shift operators

```
int a= 2;
a<<1 ;  ------------> no change
cout<< a;  ---------> 2
a= a<<1;  ----------> change ---> left shift by 1
cout <<a;  ---------> 4
```

## right shift in -ve no.

↳ link in links.txt in repo

## How -1 is stored in memory -




## ~a = -(a+1) and ~(~a) = a

ex - a = 5; ⟶ 0...0 1 0 1

a= ~a; ⟶ 1...1 0 1 0 ⟶ -6 ⟶ -(5+1)

read ↳ 2's ⟶ 0...0 1 1 0

-6

a= -6;

↳ 1...1 0 1 0

a= ~a; ⟶ 0...0 1 0 1

↳ 5 ⟶ -(-6+1)

# W2-R

## Number System

- method to represent numeric values using digits

## Decimal Number System

- base 10
  - digits → 0 to 9

## Binary Number System

- base 2
  - digits → 0, 1
- used in CPU, memory, computer
  - 0 → power off
  - 1 → power on
  - number, images, all files & folder are in binary

## Decimal to Binary

- divide no. by 2
- store remainder
- repeat above steps until no. is 0 or 1
- reverse the bits so obtained

## Binary to Decimal

- multiply each bit with its place value
  - base $^{i}$
    - $2^{i}$
- add all products

# Time & Space Complexity

W3-R

## Time Complexity

- amount of time taken by an algo as a function of length of input
- not actual time
- it defines CPU operations
- use case -
  - to make efficient programs
  - ask by interviewer after every sol. you give

## Space Complexity

- amount of space taken by an algo as a function of length of input

## Units to represent Complexity

Big O ⟶ upper bound ⟶ worst case

Theta $\Theta$ ⟶ average case

Omega $\Omega$ ⟶ lower bound ⟶ best case

## Big O Complexities

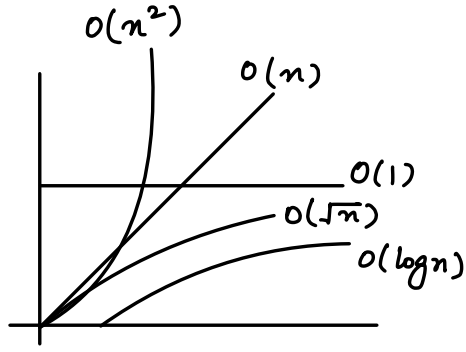
$O(1)$ ⟶ Constant time

$O(n)$ ⟶ Linear time

$O(\log_2 n)$ ⟶ Logarithmic time

$O(n^2)$ ⟶ Quadratic time

$O(n^3)$ ⟶ Cubic time




nesting ⟶ multiply ⟶

```
{
  { }
}
```

in sequence ⟶ add

```
{ }
{ }
```

$$f(n) = 4n^4 + \frac{n^3}{5} + \log n + n\log n \longrightarrow O(n^4)$$

## Complexity Order

$$O(1) < O(\log_2 n) < O(\sqrt{n}) < O(n) < O(n\log_2 n) < O(n^2)$$

$$< O(n^3) < O(2^n) < O(n!) < O(n^n)$$

# ARRAY

W3-L1

- Data Structure to store similar items
  - same datatype
- Continuous memory location space
- use case
  - for multiple huge same kind of data

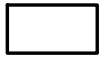
int a[30000]; ⟶ 30000 variables are ready

## continuous memory location

- memory wastage
  - if needable memory is present but not in continuous way

int a = 5;

A



a

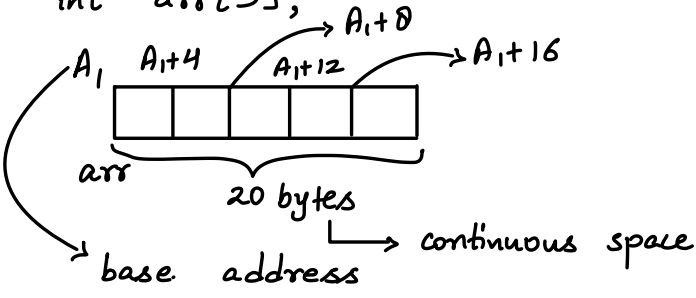
symbol table

int a ⟷ A

int arr ⟷ $A_1$

## Declaration

int arr[5];



```
cout << arr ;  ------>  A1
cout << &arr ;  ------>  A1
```

## Initialization

```
int arr[] = {2,4, 6, 8, 10};
int arr2[5] = {2, 4, 6, 8, 10};
int arr3[10] = {2, 4, 6, 8, 10};  ------> remaining 5 will be 0
//int arr4[4] = {2,4, 6,8, 10};  ------> ERROR
int arr5[10] = {0};  ------> initializing all values with 0
```

## Making array at runtime

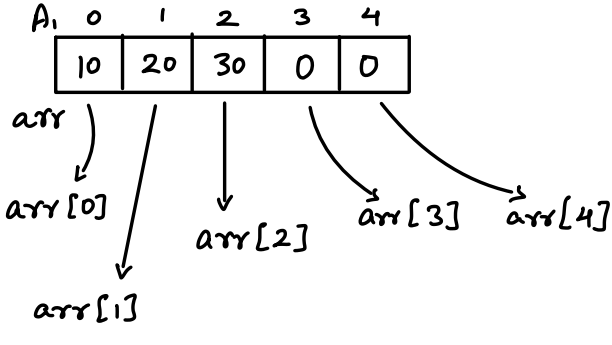
```
int n;
cin >> n;
int arr[n];  ------> BAD PRACTICE
```

## Index and Access in memory

```
int arr[5] = {10, 20, 30};
```

→ 0th based indexing

↳ 0 to n-1



arr[i] ⟶ value at address [arr + (i*4)]

- arr → $A_1$
- i → index
- 4 → due to int (datatype size)

thats why 0 based indexing → due to internal working

## taking input in array

```
cin >> arr[i];
```

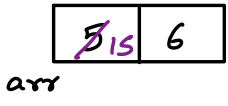
## Arrays and Function

```
func ( int arr[], int size) {
}
```

- pass by reference
- updation in actual array
- always pass size alongwith arr

```
int main() {
    int arr[] = {5, 6};
    int size = 2;
    func (arr, size);
    return 0;
}
```

```
void func ( int a[], int size){
    a[0] = a[0] +10;
}
```



sizeof(int); ⟶ 4 ⟶ in bytes

int arr[5];

sizeof(arr); ⟶ 20 ⟶ in bytes

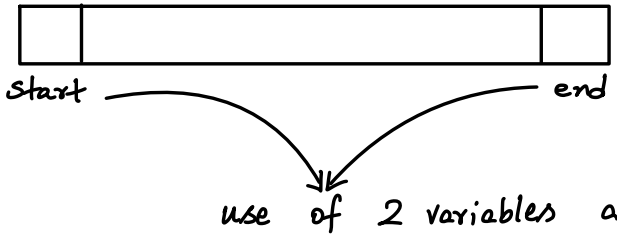
## linear search in array

## INT_MIN and INT_MAX

→ to find max., start ans with INT_MIN

→ to find min., start ans with INT_MAX

## 2 pointer approach




use of 2 variables as extreme points

## To find size of array

```
int arr[] = {1, 2, 3, 4};
int size = sizeof(arr) / sizeof(int);
```

sizeof(int) → datatype

# W3-L2

## Vector

- Data structure
- Same as array but dynamic
  - size not fixed
- default size → 0
- if gets full and new items are inserted
  size gets doubled
- pass by value in functions

## Initialization

```
vector <int> arr {10,20,30};
```
→ | 10 | 20 | 30 |

```
vector <int> arr (5);
```
→ | 0 | 0 | 0 | 0 | 0 |

```
vector <int> arr (5, -2);
```
(5 → size, -2 → value)
→ | -2 | -2 | -2 | -2 | -2 |

```
int n;
cin >> n;
```
→ let n = 5

```
vector <int> arr(n);
```
→ | 0 | 0 | 0 | 0 | 0 |

```
vector <int> arr (n, 10);
```
→ | 10 | 10 | 10 | 10 | 10 |

## Insertion -

```
arr.push_back (5);
```

## Remove

```
arr.pop_back ();
```

## Size -

```
arr.size ();
```
→ no. of elements it stores

## declaration

```
vector <int> arr;
```
- arr.size() → 0
- arr.capacity() → 0

**Empty or Not**

arr.empty(); ⟶ true if empty

**Capacity -**

arr.capacity(); ⟶ * by 2 if arr gets fully filled and a new element is inserted

- ⟶ no. of elements it can store
- ⟶ in initialization, capacity = size in all methods

**sizeof(arr);** ⟶ compiler dependent initially

cout << arr ; ⟶ give ERROR

→ Xor ⟶ Cancels out same element

0 ^ ans = ans
- ⟶ 0^1 = 1
- ⟶ 0^0 = 0

**for each loop**

```
for (auto val: arr){
    cout << val << ' ';
}
```

# 2D Arrays

# W3-L3

→ use case

→ to work on multiple rows and columns

## Declaration -

```
int arr[m][n];
```

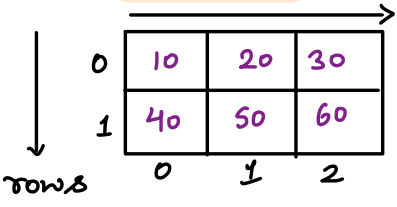
→ m*n elements

→ cols → 0 to n-1

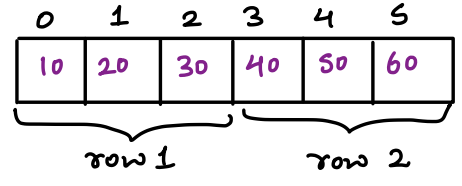
→ rows → 0 to m-1

```
int arr[2][3];
```

## Visualize -




## in memory -




## Access -

```
arr[i][j];
```

→ col index  $0 \le j < n$

→ row index  $0 \le i < m$

## Mapping -

```
linear_index = c * i + j
```

no. of cols

row index

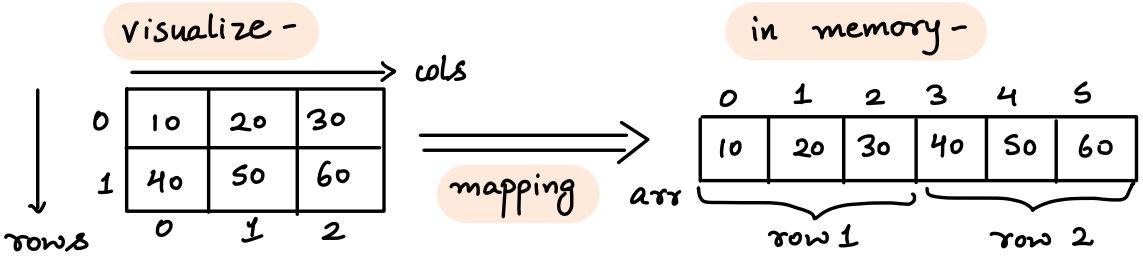
col index

$i = linear\_index / c$

$j$

## Initialization -

```
int arr[2][3] = {{10,20,30}, {40,50,60}};
```




## 2D Arrays and function -

→ pass by reference

```
func( arr[][500], int rows, int cols)
```

→ cannot leave blank

why

this value and no. of cols in array passed in func should be same

if dont know, put large value

for mapping

→ linear_index = c * i + j

→ So that compiler can know

## 2D Array -

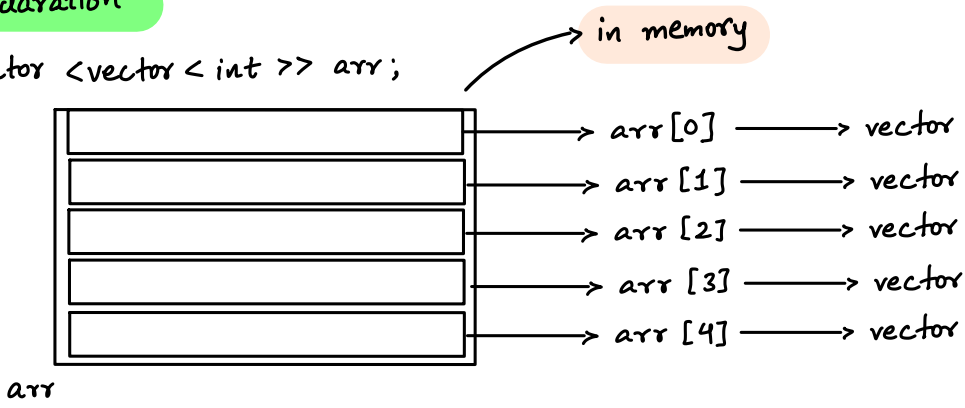
→ to make dynamically

→ vector of vectors

# 2D Vector

## Declaration -

```
vector <vector < int >> arr ;
```




## Declaration -

```
vector < vector <int >> arr ;
vector < vector <int >> arr (n);
```

n → no. of rows

## Number of rows

```
arr. size ()
```

## Number of cols

```
arr [i]. size ()
```

- → in ith row
- → size of ith row

## Initialization

```
vector < vector <int > > arr ;
vector < vector < int > > arr ( rows , ( vector <int > (cols , value ));
```

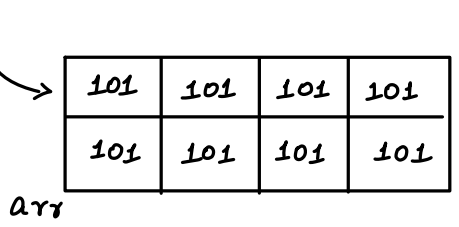
size ↑ rows

value ↑ vector<int>(cols, value)

initialization of 1D vectors in arr

- rows → no. of rows in arr
- cols → no. of cols in arr
  - → size of 1D arrays
- value → initial value in all elements of all 1D vectors

```
vector<vector<int>> arr(2, vector<int>(4, 101));
```



# Searching and Sorting

W4-L1

## Searching

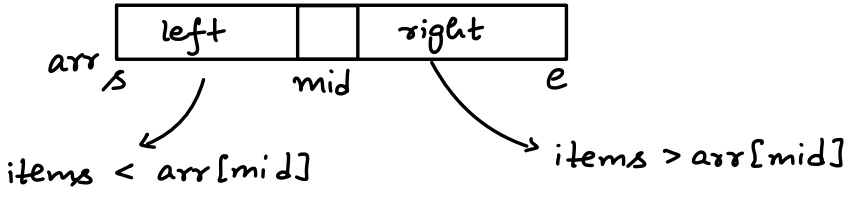
### Linear Search

```
int linearSearch ( vector <int> arr, int target ) {
    int n = arr.size();
    for( int i=0; i<n ; i++){
        if ( target == arr[i])
            return i;
    }
    return -1;
}
```

T.C - O(n)

### Binary Search -

- condition → sorted order → monotonic function
- binary → 2 → start and end pointers



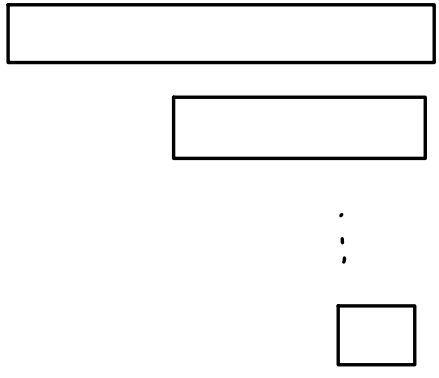
```
int binarySearch ( int arr[], int n , int target ){
    int s= 0, e= n-1;
    int mid = s+(e-s)/2;
    while( s<=e) {
        int element = arr[mid];
        if (target == element)
            return mid;
        else if ( target < element)
            e = mid - 1; ----> search in left subarray
        else
            s = mid +1; ----> search in right subarray
        mid = s + (e-s)/2;
    }
    return -1;
}
```

issue in $mid = \frac{s+e}{2}$; ⟶ int overflow if $s+e < INT\_MAX$

⟶ so use $mid = s + \frac{e-s}{2}$;

T.C - $O(\log_2 n)$

## T.C. of binary Search




at last $\frac{n}{2^k} = 1$

$k = \log_2 n$

So loop runs $\log_2 n$ times

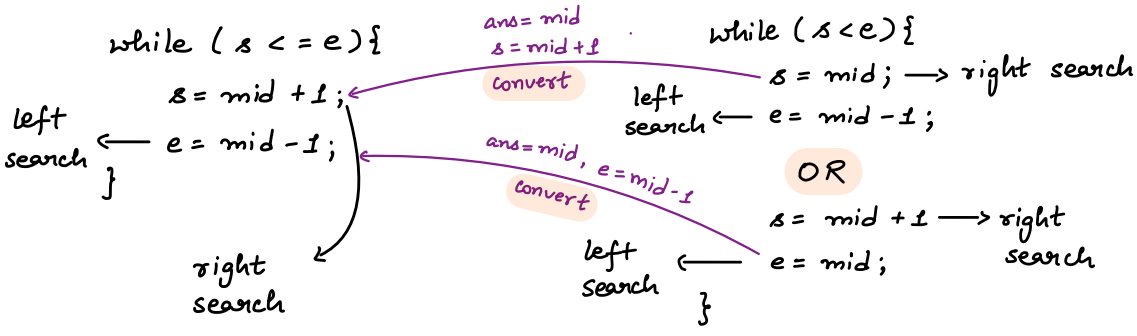
T.C. = $O(\log_2 n)$

## W4-L2

→ Search Space

- → find range of search space (start & end) in ques of. Binary Search Questions
- → store mid in ans if needed

→ In binary search questions




# W4-L3

cout << (int)(-3.74);

↳ -[int(3.74)]

↳ -(3) → -3

cout << (-22)/7; → -3

cout << 22/(-7); → -3

## Types of ques in binary search

→ 1st type → classic questions

↳ lower bound

upper bound

peak in mountain array

pivot in sorted rotated array ← can also find array is sorted or not

↳ pivot index = n-1

search in sorted rotated array

→ 2nd type → find in search space (range)

- → predicate function → logic to decide either left or right
- → sqrt of a no.
  divide 2 numbers

Advance Binary Search Problems

- → Book allocation
  Painters Partition
  Aggressive Cows
  Roti / Paratha Spoj
  Eko Spoj

→ 3rd type → observation in index

- → missing element in sorted array
  odd appearing element in array

W4-L5

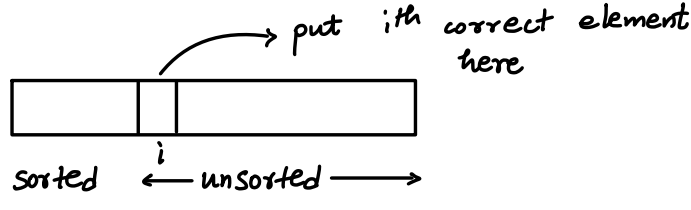
Sorting

- → putting all elements either in increasing order or decreasing order

## Selection Sort -

- select minimum element and put it in its right position
- select correct element for $i^{th}$ index
  - minimum

```
void selectionSort ( int arr[], int n) {
    for(int i=0; i<n-1; i++){
        mini = i;
        for ( int j= i+1 ; j<n ; j++) {
            if ( arr[mini] > arr[j])
                mini = j;
        }
        swap (arr[i], arr[mini]);
    }
}
```




T.C. $O(n^2)$

S.C. $O(1)$

Use Case - for small size array

## Bubble Sort -

→ in $i^{th}$ round put $i^{th}$ largest element to its correct position using adjacent comparisions

```
void bubbleSort ( int arr[], int n){
    for(int i=0 ; i<n-1 ; i++) {
        bool swapping = false;
        for( int j= 0 ; j<n-i-1 ; j++){
            if (arr[j] > arr[j+1]) {
                swap ( arr[j], arr[j+1]);
                swapping = true ;
            }
        }
        if ( swapping = = false)
            break;
    }
}
```

T.C. - $O(n^2)$ → worst and average case (→ reverse sorted)

$O(n)$ → best case → already sorted

S.C. - $O(1)$

Use Case - To put $i^{th}$ largest element to its correct position

## Insertion Sort -

↳ take an element and insert it on its correct position by shifting

```
void insertionSort ( int arr [ ], int n ) {

    for ( int i = 1 ; i < n ; i++ ) {
        int curr = arr [ i ];
        int j = i - 1 ;
        for ( ; j >= 0 ; j-- ) {
            if ( arr [ j ] > curr )
                arr [ j+1 ] = arr [ j ] ; // shifting
            else
                break ;
        }
        arr [ j+1 ] = curr ; // inserting
    }
}
```

T.C. - $O(n^2)$ ⟶ worst & average case

$O(n)$ ⟶ best case

S.C. - $O(1)$

Use Case - When array is small or when array is partially sorted

## Inbuilt sort function

- sort (arr.begin(), arr.end());
- algo used is Intro sort
  hybrid of quick sort, heap sort, insertion sort
- min. time then any of other sort

T.C. - O(nlogn)

S.C. - Not Defined

## Stable and Unstable Algorithm

Stable → order preserve after sorting
→ of same values

Stable Sorting Algo

2 1 2 2 3

↓ after sorting

1 2 2 2 3

if A[i] = A[j] , i < j

then A[i] comes first before A[j] after sorting too

Stable - Bubbe Sort, Insertion Sort, Merge Sort, Count Sort

Other (unstable) sorting algorithms can be made stable by some changes

# T.C. and S.C Sorting Algorithm Table

| Algo | S.C. | Worst T.C. | Avg. T.C. | Best T.C. |
|---|---|---|---|---|
| Selection | $O(1)$ | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Bubble | $O(1)$ | $O(n^2)$ | $O(n^2)$ | $O(n)$ |
| Insertion | $O(1)$ | $O(n^2)$ | $O(n^2)$ | $O(n)$ |
| sort func. (intro sort) | not defined | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |
| | | | | |

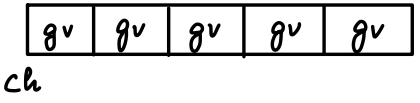
# CHAR ARRAYS & STRING

W5-L1

## Char Arrays -

- Data structure ⟶ used to store data
- Not Data type ⟶ tells type of data

```
char ch[5];
```




## Taking input in char array

```
char ch[7];
cin >> ch[i];
```

`cin >> ch;` ⟶ by defaut NULL char will insert at end ⟶ '\0'

↓

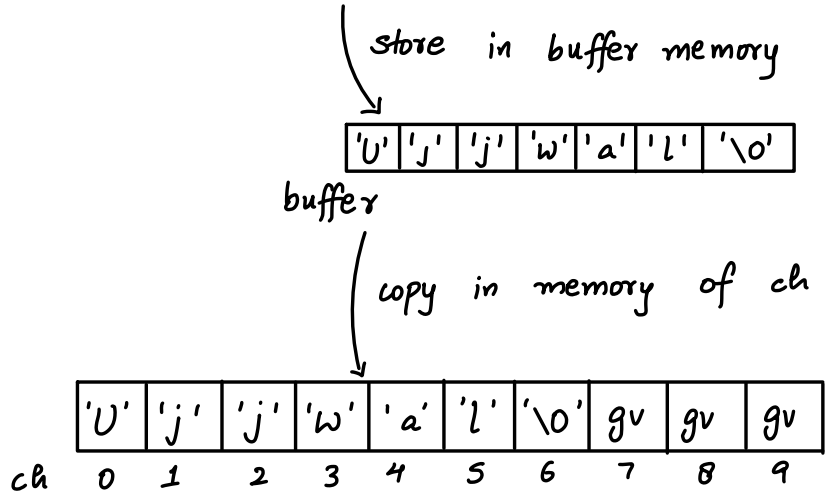
NULL char shows string termination

- cin reads until it gets any white space
  - space ' '
  - tab '\t'
  - endl '\n'

`cout << ch;` ⟶ print until it gets delimeter

```
char ch[10];
```

```
cin >> ch;  ⟶  Ujjwal
```




```
cout << ch;  ⟶  Ujjwal  ⟶  stops after 'l'
```

because of '\0' char

## Overflow -

```
char ch[4];
cin >> ch;  ⟶  Ujjwal
cout << ch;  ⟶  Compiler Dependent
```

## sizeof (char array)

```
char ch[] = "Ujjwal";
cout << sizeof(ch);  ⟶  7
                        ⟶  6 + 1
                            ⟶  NULL char
```

## get line

cin.getline ( char array, max char to write, delimeter);

char where taking input stops

by default '\n' ⟶ enter

ex- cin.getline ( ch , 50);

cin.getline (ch , 50, ' ');

## Char arrays and function

↳ pass by reference

```
func ( char ch[]){

}
```

## size of char array ⟶ sizeof (ch) / sizeof (char)

sizeof (char) ⟶ 1

## Some inbuilt functions of char array

strlen ( ch );

strcmp (ch1, ch2);

strcpy ( ch1 , ch2);

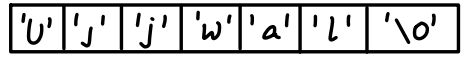
## Strings

- Datatype
  Not DataStructure
- Dynamic char array
- NULL char at last of string

string str; ⟶ empty string created

cin >> str; ⟶ Ujjwal

cout << str; ⟶ Ujjwal




str

## getline -

```
string str;
getline (cin, str);
```

## char array

```
char ch[100] = "B_abba_r";
cout << ch;
```
⟶ B_abba_r

```
ch[1] = '\0';
ch[6] = '\0';
cout ch;
```
⟶ B

↓

stops just as it gets NULL char

## string

```
string str = "B_abba_r";
cout << str;
```
↳ B_abba_r

```
str[1] = '\0';
str[6] = '\0';
cout << str;
```
↳ Babbar

↓

Runs till the length of string

## Sort function in strings -

W5-L3

→ works on randomised quick sort

```
string str = "babbar";
sort(str.begin(), str.end()); ----> aabbbr
sort(str.begin(), str.end(), greater<char>());
                                  ----> rbbbaa
```

## Custom Comparator -

```
bool cmp(char a, char b){
    return a<b;  ----------> can be any function according to need
}
bool cmp2(char a, char b){
    return a>b;
}
int main(){
    string str = "babbar";
    sort(str.begin(), str.end(), cmp);
    cout << str; ----------> aabbbr
    string str2 = "babbar";
    sort(str.begin(), str.end(), cmp2);
    cout << str2; ----------> rbbbaa
    return 0;
}
```

## Hash Map → WILL LEARN LATER

- Data structure
- data stored in key-value pair

## Initialization

```
map<key datatype, value datatype> map_name;

map<int, char> m; ——> ordered map

m[0] = 'a';

m[1] = 'b';

m[25] = 'z';

cout << m[0]; ——> 'a'

cout << m[25]; ——> 'z'

cout << m[20]; ——> '' ——> NULL char

cout << (int) m[20]; ——> 0
```

## auto keyword

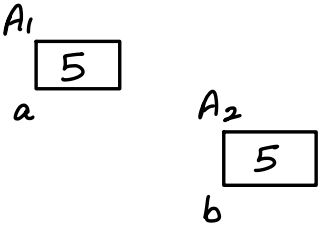
- finds automatically datatype of variable

# SYMBOL TABLE

W6-L1

- Data Structure
- Stores mapping of variable name and memory location address → Done by OS
- entries in symbol table cant be change

```
int a = 5;
int b = 5;
```




Symbol Table

| name | address |
|---|---|
| a | $A_1$ |
| b | $A_2$ |

→ entry of a in symbol table is made

```
cout << a;
```
→ 5

## & Address Of Operator

```
cout << &a;
```
→ $A_1$ → address of variable a in memory (hexadecimal)

```
cout << &b;
```
→ $A_2$

$A_2 = A_1 \pm 4$ → because of consecutive variables in memory

↓ int datatype of a and b

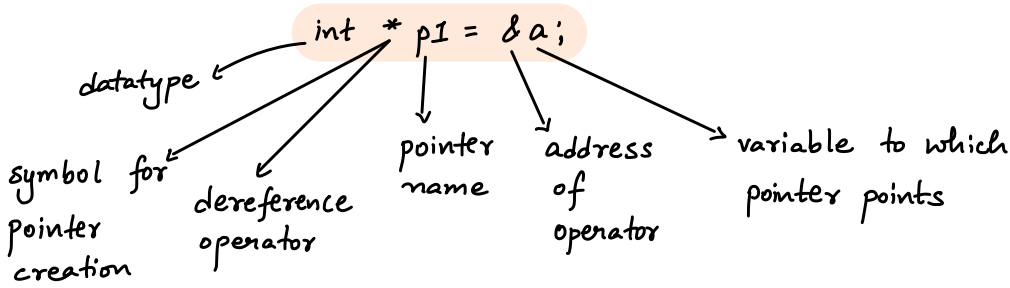
# POINTERS

→ stores address

→ NOT a datatype, just a variable storing address of another variable

```
int a = 5;
int * p1 = &a;
```

p1 is a pointer to integer datatype



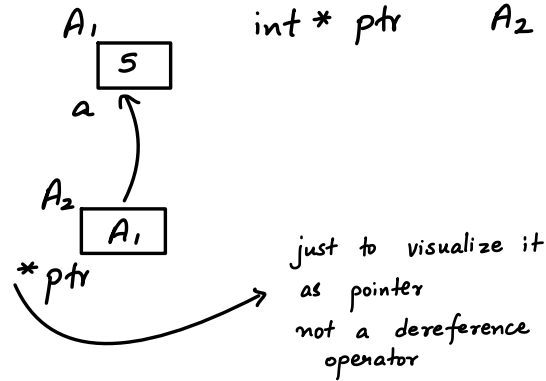

```
string s = "Ujjwal 2327";
string * ptr = &s
```

→ ptr is a pointer to string datatype

```
int a=5;
int * ptr = &a;
cout << a;        ⟶ 5
cout << *a;       ⟶ ERROR
cout << &a;       ⟶ A1
cout << ptr;      ⟶ A1
cout << *ptr;     ⟶ 5
cout << &ptr;     ⟶ A2
```

symbol table

| | | |
|---|---|---|
| int | a | $A_1$ |
| int * | ptr | $A_2$ |



*ptr ⟶ value at (address stored in ptr)

↳ dereference operator

&ptr ⟶ address of ptr

## size of pointer

sizeof(ptr); ⟶ 8 ⟶ Always

architecture dependent
compiler implementation
memory organization

64 bit architecture

it stores address,
datatype does not matters

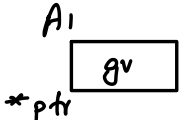
## Use case of pointer

- dynamic memory allocation
- memory managment
- pointer arithmetic
  - ↳ go from one location to other
- pass by reference in array
- to create pointer to function
  - ↳ passing a function inside another function as an argument

int * ptr; ⟶ VERY BAD PRACTICE

⟶ Segmentation fault

cout << ptr; ⟶ gv



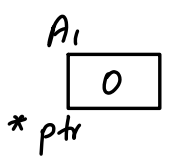
ptr points to a memory location that may not be of its program

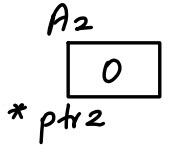
**Segmentation fault -**

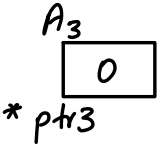
→ using other's memory

## NULL Pointer

```
{int * ptr = 0;
{ int * ptr2 = NULL;
{ int * ptr3 = nullptr;
  cout << ptr ;  ——→ 0
  cout << *ptr ;  ——→ ERROR
  cout << &ptr ;  ——→ A1
```



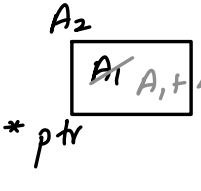




→ Segmentation fault

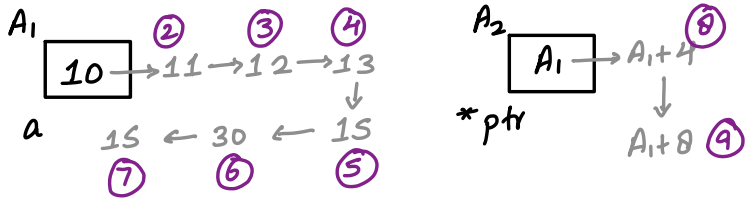
## Arithmetic In Pointers

```
int  a = 5;
a++;
int * ptr = &a;
ptr++;
*ptr++;
```


→ $A_1$ to $A_1+3$ has already be taken by integer a, so next address will be $A_1+4$

```
int  a = 10;
int *ptr = &a;
```



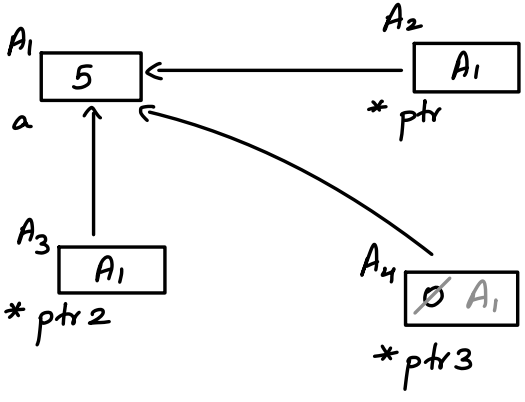

| | |
|---|---|
| a | 10 |
| &a | A1 |
| ptr | A1 |
| *ptr | 10 |
| &ptr | A2 |

| | | |
|---|---|---|
| 1 | *ptr * 2 | 20 |
| 2 | (*ptr)++ | 10 |
| 3 | ++(*ptr) | 12 |
| 4 | a = a+1 | 13 |
| 5 | *ptr = *ptr + 2 | 15 |
| 6 | *ptr = *ptr * 2 | 30 |
| 7 | *ptr = $\frac{*ptr}{2}$ | 15 |
| 8 | *(ptr++) | 15 |
| 9 | *(++ptr) | gv |

```
int  a = 5;
int *ptr = a;   ⟶ ERROR
```

## Copying a pointer

```
int  a = 5;
int *ptr = &a;
{ int *ptr2 = ptr;
{ int *ptr3 = 0;
  ptr3 = ptr;
```



# ARRAYS & POINTERS

W6-L2

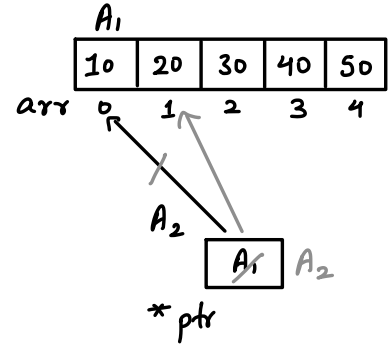
```
int arr[] = {10, 20, 30, 40, 50};
```

arr → CONSTANT pointer to the first element of array

- cannot change
- also does not have seperate memory

arr[i] ⟶ element at index i

& arr[i] ⟶ address of arr[i]




```
cout << arr[0];   ⟶ 10
cout << &arr[0];  ⟶ A1
```

```
cout << arr;   ⟶ A1
```
⟶ array name gives Base Address

```
cout << &arr;  ⟶ A1
```
⟶ from Symbol table

Same address unlike pointers

```
int *ptr = arr;
cout << *ptr;   ⟶ 10
cout << ptr;    ⟶ A1
cout << &ptr;   ⟶ A2

cout << *arr;       ⟶ 10
cout << *(&arr);    ⟶ 10
cout << *(&arr[0]); ⟶ 10
```

Symbol table

| type | name | address |
|---|---|---|
| int(*)[5] | arr | A1 |
| int * | ptr | A2 |

cout << *arr +1; ⟶ 11

cout << *(arr + 1); ⟶ 20

cout << *(arr + 2); ⟶ 30

cout << arr[2]; ⟶ 30

cout << 2[arr]; ⟶ 30

arr[i] ⟷ *(arr + i) ⟷ i[arr]

arr++; ⟶ ERROR ⟶ Entry in symbol table cant be change

ptr++; ⟶ A1+4

→ You can access subpart of an array using pointers

cout << *(ptr + 2); ⟶ 40

cout << *(ptr +100); ⟶ gv / segmentation fault / out of bound error

**Arrays / Array pointers**

int arr[] = {10, 20, 30};

① cout << arr; ⟶ A1
cout << &arr; ⟶ A1
} Same

② arr++; ⟶ ERROR

③ sizeof(arr); ⟶ 3*4=12

**Pointers / Normal pointers**

int *ptr = arr;

① cout << ptr; ⟶ A1
cout << &ptr ⟶ A4
} diff.

② ptr++; ⟶ VALID

③ sizeof(ptr) ⟶ 8
↓
size of address

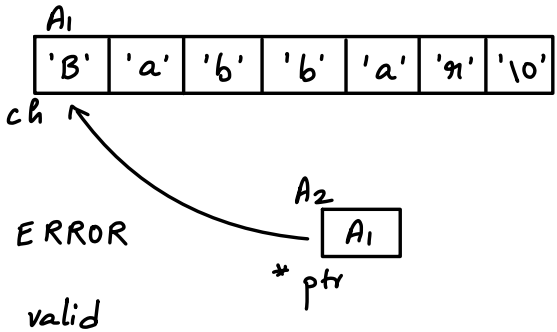
```
int * p1 = arr;
int * p2 = &arr;
int * p3 = &arr[0];
```

→ ERROR

Why?

## CHAR ARRAYS & POINTERS

```
char ch[] = "Babbar";
char * ptr = ch;
char * p = &ch;
char * p2 = &ch[0];
```




→ ERROR

→ valid

```
cout << ch;
```

→ Babbar
→ not an address

```
cout << ptr;
```

→ Babbar
→ not an address

whole string from that location until NULL char

cout implementation is diff. in char pointers and char arrays

```
cout << &ch;    → A1
cout << ch[0];  → B
cout << &ch[0]; → Babbar
cout << *ptr;   → B
cout << &ptr;   → A2
```

ch[i] ⟷ *(ch + i) ⟷ i[ch]

→ ch, &ch[0], and ptr values are addresses but due to diff. cout implementation in char pointers and char arrays,
cout << ch , cout << &ch[0] and cout << ptr will give Babbar

```
char ch[] = "Sherbano";
char *ptr = ch;
```

cout << ch ; ⟶ Sherbano

cout << *ch ; ⟶ S

cout << &ch ; ⟶ $A_1$

cout << *(ch + 3); ⟶ n

cout << ptr ; ⟶ Sherbano

cout << &ptr ; ⟶ $A_2$

cout << *(ptr + 3); ⟶ n

cout << ptr + 2; ⟶ erbano

cout << *ptr ; ⟶ S

cout << ptr + 8; ⟶ " ⟶ NULL char

cout << ptr + 9; ⟶ gv

```
cout << ch[0];      ——————→ S
cout << &ch[0];     ——————→ Sherbano
cout << &(*ch);     ——————→ Sherbano
```

## CHAR AND POINTER

```
char ch = 'k';
char * ptr = &ch;

cout << ch;      ——————→ k
cout << &ch;     ——————→ k.....
                          → gv
                          → print until it gets '\0'
cout << ptr;     ——————→ k.....
                          → gv
                          → print until it gets '\0'
cout << &ptr;    ——————→ A2
cout << *ptr;    ——————→ k
```

## Behind The Scenes

```
char ch[10] = "Babbar";
```

→ 2 step process

① temp memory

| 'B' | 'a' | 'b' | 'b' | 'a' | 'r' | '\0' |
|---|---|---|---|---|---|---|

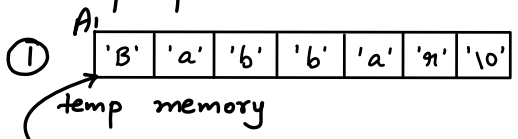
copy ↓

② ch

| 'B' | 'a' | 'b' | 'b' | 'a' | 'r' | '\0' | gv | gv | gv |
|---|---|---|---|---|---|---|---|---|---|

```
char * ch = "Babbar"
```

→ 2 step process




② A1 → pointer points to temporary memory

* ch

↓

BAD PRACTICE

## POINTERS WITH FUNCTIONS

→ pass by value

a copy of pointer is made

→ but simulates pass by reference

↳ as in case of arrays & functions

pointer is passed

```
int func ( int arr[]){
    cout << arr;      ——→ A1
    cout << *arr;     ——→ 10
    cout << &arr;     ——→ A2
    cout << sizeof(arr); ——→ 8
}
```

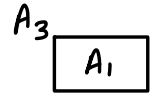
A2 | A1

* arr

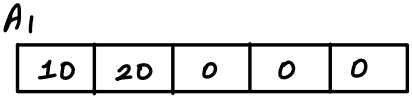
copy of pointer is created

```
int func2 ( int * arr ){
    cout << arr ;  ------------------> A1
    cout << * arr ;  ----------------> 10
    cout << & arr ;  ----------------> A3
    cout << sizeof ( arr ) ;  -------> 8
}
```

pointer is passed




copy of pointer is created




```
int main ( ) {
    int arr [5] = {10, 20};
    cout << arr ;  ------------------> A1
    cout << * arr ;  ----------------> 10
    cout << & arr ;  ----------------> A1
    cout << sizeof ( arr ) ;  -------> 5*4 = 20
    func ( arr );
    func ( arr 2 );
    return 0;
}
```

arr

whole array will not pass

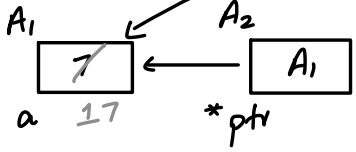
only array pointer / base address will pass

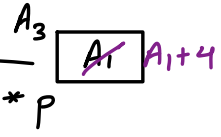
pass by reference

copy of pointer pointing to base address will be created

```
int main () {
    int a = 7;
    int * ptr = & a ;
    update (ptr);
    return 0;
}
```

```
void update ( int * p ) {
    (1) *p = *p + 10;
    (2) p++;
}
```




*(A1) = *(A1) + 10

*(A1) = 7 + 10 = 17

a = 17

Function to Pointers in HW

Basic Mathematics for DSA

# W6-L3 R

Sieve of Eratosthenes Theorem

→ to find no. of prime numbers between 1 & n

Steps- → seive

- make an array of size n and mark them all as primes
- Start from 2 till end, mark all no. > 2 comes in the table of 2 as non prime
- Do above step for numbers 2 to <n if they are marked prime
- Count all remaing marked prime numbers

```
int countPrimes(int n) {
    if (n <= 1)
        return 0;

    vector<int> isPrime(n, true);
    isPrime[0] = isPrime[1] = 0;   // making both 0

    int ans = 0;
    for(int i=2; i<n; i++){
        if (isPrime[i]){
            ans++;
            for(int j= 2*i; j<n; j+=i)
                isPrime[j] = false;
        }
    }
    return ans;
}
```

`i*i < n` → to optimize T.C. as 2i to (i-1)*i are already marked when i= 2 to (i-1)

TC - $O(n * \log(\log n))$ SC - $O(n)$

$$n\left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \frac{n}{11} + \cdots\right]$$

$$n^2\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{11} + \cdots\right]$$

$$n \log(\log n)$$

Same with segmented sieve ⟶ having high & low

# GCD/HCF and LCM

- Greatest Common Divisor
- gcd(a, b) = gcd(a-b, b) a > b

  gcd(a, b) = gcd(b-a, a) b > a

  gcd(a, b) = gcd(a % b, b) a > b

  gcd(a, b) = gcd(b % a, a) b > a

  Use first method as % is heavy operator and computer takes more time
- Euclid Algorithm

  Apply above formula till one of the parameter becomes 0
  And other one will be GCD

```
int gcd( int a, int b) {
    if (a==0)
        return b;
    else if (b==0)
        return a;
    while (a>0 && b>0){
        if (a>b)
            a=a-b;
        else
            b=b-a;
    }
```

```
    return (a == 0) ? b : a;
}
```

TC - O( min(a, b))

## LCM

$$lcm * gcd = a * b$$

$$lcm = \frac{a * b}{gcd}$$

## Modulo Arithmetic

- $a \% n \longrightarrow [0, n)$
- $(a + b) \% n = (a \% n) + (b \% n)$

$$(a - b) \% n = (a \% n) - (b \% n)$$

$$(a * b) \% n = (a \% n) * (b \% n)$$

$$(\ldots((a \% n) \% n) \ldots \% n) = a \% n$$

## Fast Exponentiation

- $a^b = a^{b/2} * a^{b/2}$ , b is even
- $a^b = [a^{b/2} * a^{b/2}] * a$ , b is odd

```
int fastExponentiation ( int a, int b){
    int ans = 1;
    while (b){
        if (b & 1)
            ans = ans * a;
        a = a* a ;
        b >>= 1 ;   // b = b >>1  or  b = b/2
    }
    return ans
}
```

dry run code on $2^5$, if confusion

T.C. - O(log n)

Learn wild, void and dangling pointers from dashboard after learning dynamic allocation

# MULTI LEVEL POINTER

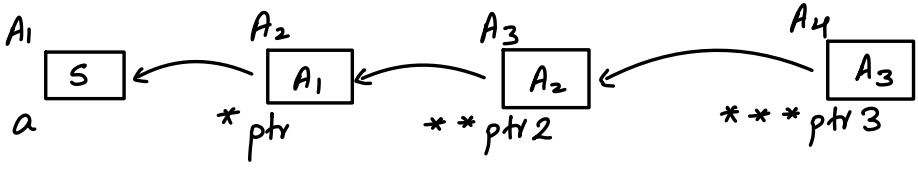
W6-L4

```
int a = 5;
int *ptr = &a;
int **ptr2 = &ptr ⟶ double pointer
```

↳ ptr2 is a pointer to int * data

```
int ***ptr3 = &ptr2;
```

↳ ptr3 is a pointer to int ** data




a ⟶ 5

&a ⟶ A1

&&a ⟶ ERROR

ptr ⟶ A1

&ptr ⟶ A2

*ptr ⟶ 5

ptr2 ⟶ A2

&ptr2 ⟶ A3

*ptr2 ⟶ A1

**ptr2 ⟶ 5

apply dereference operator 2 times

ptr3 ⟶ A3

&ptr3 ⟶ A4

*ptr3 ⟶ A2

**ptr3 ⟶ A1

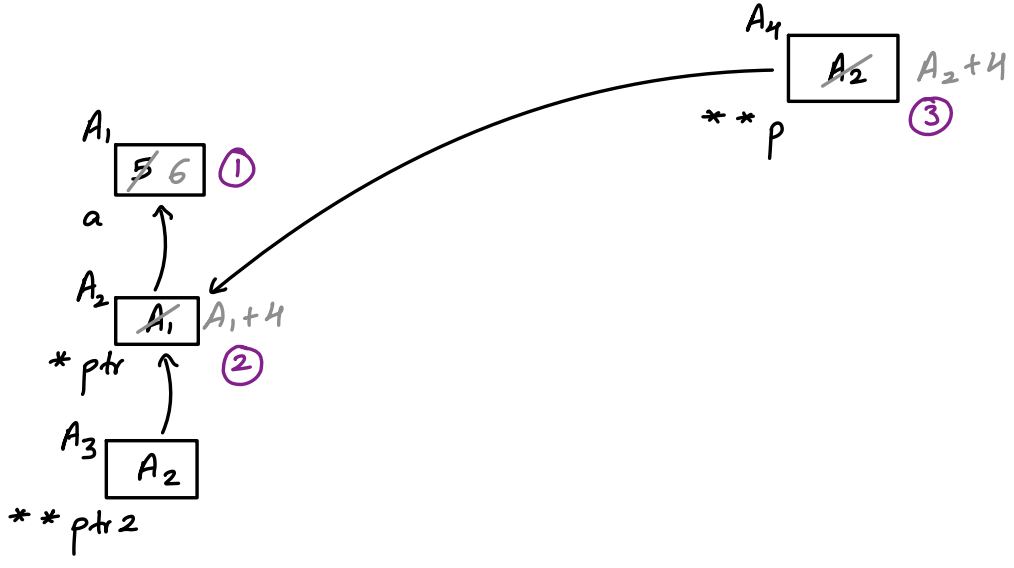
***ptr3 ⟶ 5

```
int main(){
    int a = 5;
    int *ptr = &a;
    int **ptr2 = &ptr;
    func(ptr2);
    return 0;
}
```

both are same

```
func(&ptr);
```

```
void func(int **p){
① (**p)++;
② (*p)++;
③ p++;
}
```



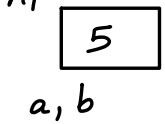

```
int a = 5;
int *p = &a;
int **q = p;
```

→ ERROR

# REFERENCE VARIABLE

- alternate of pointers → a, *ptr, **ptr → a
  - very confusing
- diff. names of same variable
  - same memory location
- only new entry in symbol table, no other memory will allocate
- can access a variable by diff. names

```
int  a = 5;
int  &b = a;
```




Symbol Table

| | |
|---|---|
| a | $A_1$ |
| b | $A_1$ |

## Use Case

- reference variable cant be set to NULL
  - pointers can be set to NULL
  - So more safety in reference variable
  - Always points to valid object / variable
- pointers are difficult to understand
  - more readability in reference variable
- generally used to implement PASS BY REFERENCE concept

# PASS BY REFERENCE

- reference variable passes in function
- does not create copy

```
int main () {
    int a = 5;
    update (a);
    update2 (&a);
    int *ptr = &a;
    //update3 (&a);
    update3 (ptr);
    return 0;
}
```

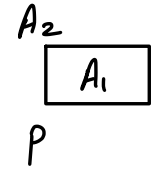
ERROR constant add. in pass by reference

```
void update( int &x){
① x++;
}
```

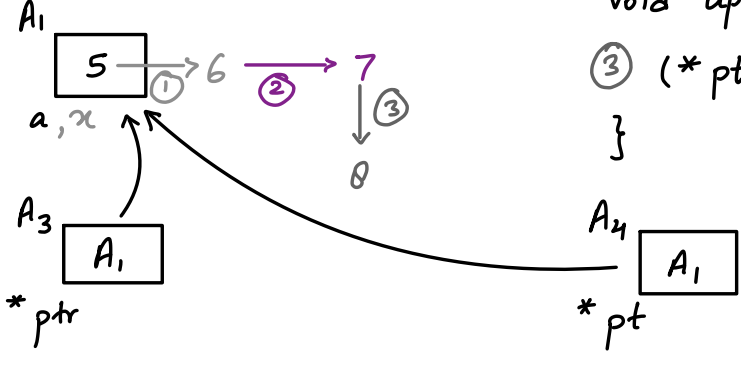
PASS BY REFERENCE

```
void update2 ( int * p) {
② (*p)++;
}
```

PASS BY VALUE




```
void update3 (int * &pt){
③ (*pt)++;
}
```



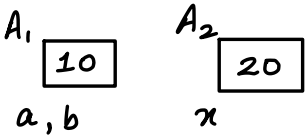
# RETURN BY REFERENCE

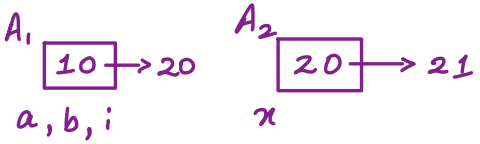
- return a variable, not a value
- To implement atleast 1 of 2 conditions must be true.
  1. passing a reference variable
  2. passing a global variable

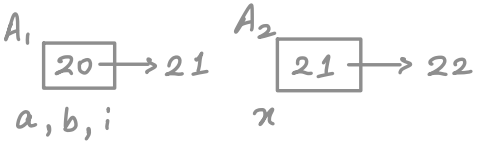
## passing a reference variable

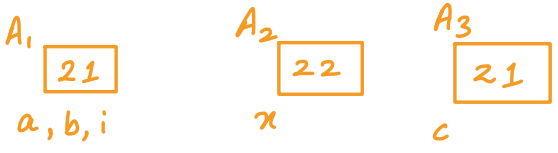
```
int main (){
    int a = 10;
    int &b = a;
    int x = 20;
    ref(a) = x;
    x++;
    ref (b) = x;
    x++;
    int c = ref(a);
    int &d = ref(a);
    return 0;
}
```

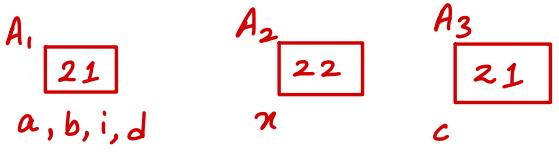
```
int & ref( int &i) {
    return i;
}
```

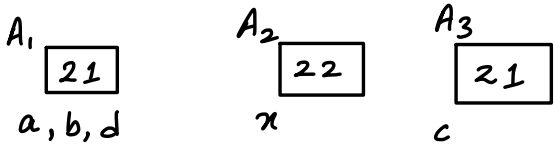

A1
10
a, b
A2
20
x


A1
10 → 20
a, b, i
A2
20 → 21
x


A1
20 → 21
a, b, i
A2
21 → 22
x


A1
21
a, b, i
A2
22
x
A3
21
c


A1
21
a, b, i, d
A2
22
x
A3
21
c


A1
21
a, b, d
A2
22
x
A3
21
c

## passing a global variable

```
int x = 5;

int main () {
    ref () = 10;
    ref 2 () = 20;
    return 0;
}
```

```
int & ref () {
    return x;
}
```

x = 10;

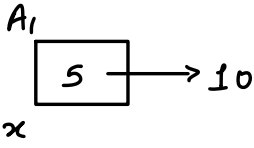
```
int ref 2 () {
    return x;
}
```

ref 2 () = 20; ⟶ ERROR

returning a value, not variable

10 = 20;

⟶ that's why ERROR




## pass by value but return by reference

```
int main () {
    int a = 10;
    int & b = a;
    int x = 20;
    ref (a) = x;
    x ++;
    ref (b) = x;
    x ++;
```

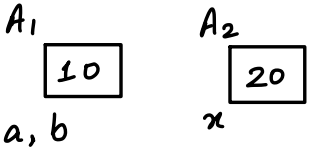
BAD PRACTICE ⟶ ERROR

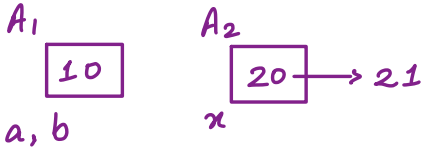
```
int & ref ( int temp ) {
    return temp ;
}
```

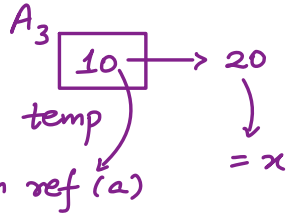
returning a variable stored in temporary memory

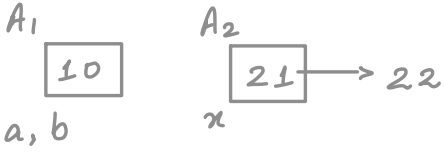
```
    int c = ref(a);
    int &d = ref(a);
    return 0;
}
```

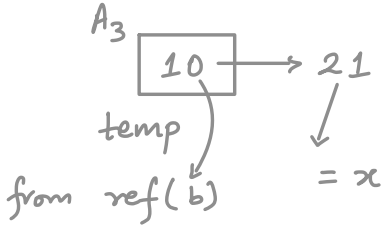
lets assume it will learn, then this would happen

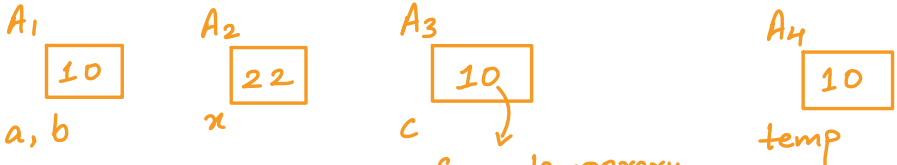


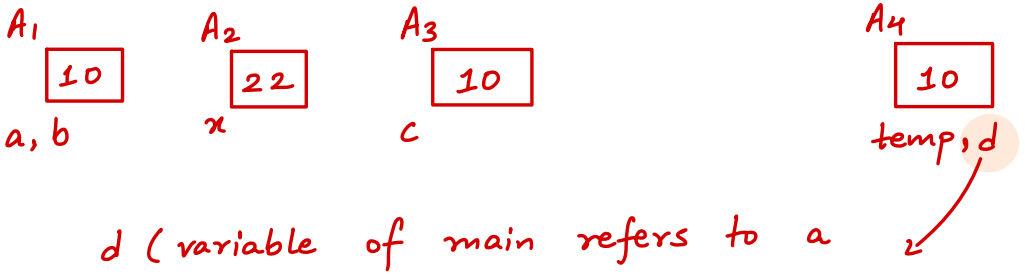




from temporary memory → whose value can be change before assignment

BAD PRACTICE ←

suppose there is no change in value of temp variable ( temporary memory , and move furthur




d ( variable of main refers to a temporary memory )

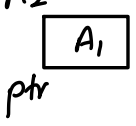
→ BAD PRACTICE

DON'T DO THIS AT HOME

Have similarity

IMP QUESTION

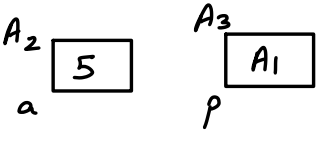
```
int main () {
    int * ptr = solve ();
    return 0;
}
```




→ pointing to a temp memory , 'a' variable will finished outside solve function

→ BAD PRACTICE

```
int * solve () {
    int a = 5;
    int * p = &a;
    return p;
}
```



# Some Important Questions

W6-A

→
```
int *ptr = 0;
int a = 10;
*ptr = a;        ——→ runtime ERROR
cout << *ptr ;
```

→
```
char * ch = 'a';
char * ptr = &a;
ch++;
cout << *ptr ;   ——→ 'b'
```

→
```
int a[] = {1, 2, 3, 4, 5}
int * p = a++;   ——→ ERROR
cout << * p ;
```

- pointer of static variables are constant pointers
- pointer of dynamic memory allocation variables can be change

name of array refers to base address

→
```
char ch = "hello";
cout << ch ;     ——→ hello
ch++;            ——→ ERROR
cout << ch;
```

→ `double arr[] = {2.5, 7.9, 50.25};`

`double * ptr = arr;`

`ptr = ptr + 0.25;` ⟶ ERROR

`cout << ptr;`

↳ pointer arithmetic op. can only be done using integers

→ `int arr[] = {1, 2, 3, 4, 5};`

`int * ptr1 = arr;`

`int * ptr2 = arr + 3;`

`cout << ptr2 - ptr1;` ⟶ 3, NOT 12

→ `int a = 5;`

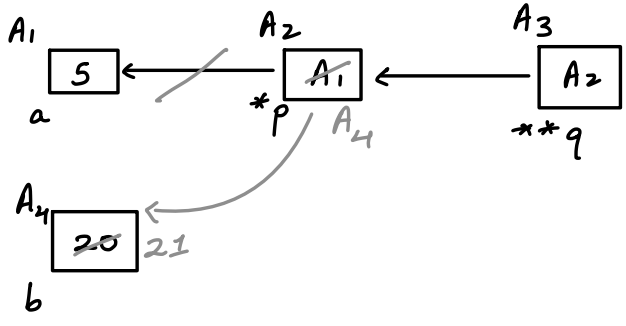
`int * ptr = &a;`

`int ** ptr2 = &ptr;`

`ptr2 = &a;` ⟶ ERROR ⟶ ** int cant convert to * int

`cout << *ptr2;`

→ `int a = 5;`
`int * p = &a;`
`int ** q = &p;`
`int b = 20;`
`* q = &b;`
`(*p)++;`
`cout << a << ' ' << b;` ⟶ 5 21



```
int a = 5;
int * ptr = &a;
int ** ptr2 = &ptr;
int ** ptr3 = &(&a);
cout << ** ptr3;
```

→ ERROR

↓

address of (address in memory location)

(doesn't make any sense)

W6-L5

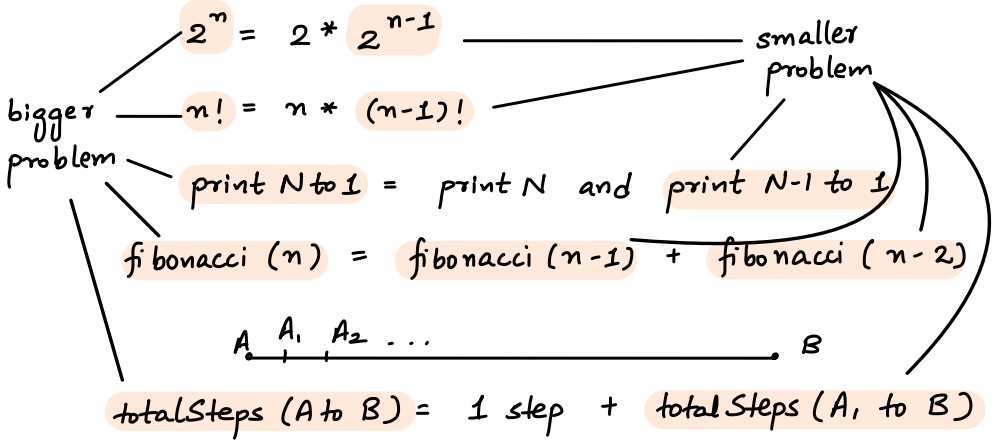
POITER IMP DOUBD

(IN DASH BOARD)

# RECURSION

W7-L1

- When a function calls itself
- When sol. of bigger problem depends on similar smaller problem(s)




## Components of Recursion

- Base Case → stopping cond. (return statement) always first
- Recurrence Relation (recursive call)
- Processing → optional

## Head Recursion

- recurrence relation before processing

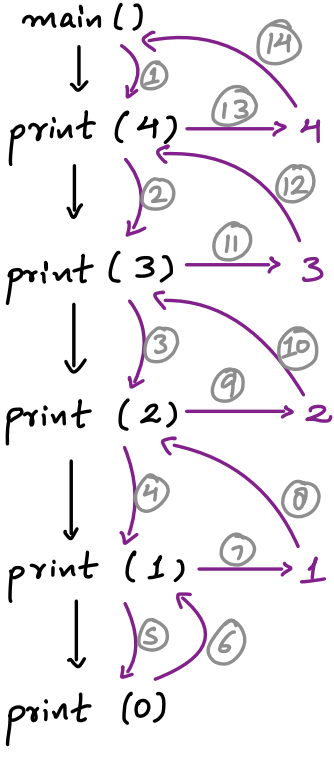
## Tail Recursion

- recurrence relation after processing

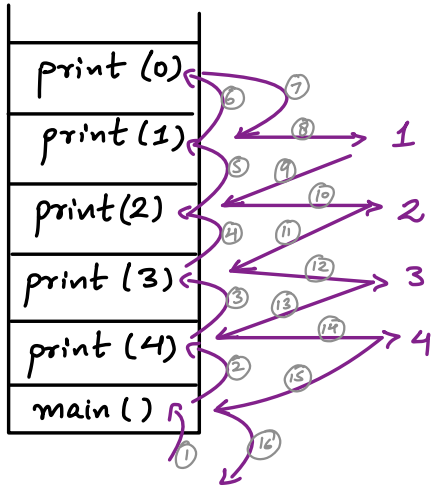
```
int main () {
    int n = 4;
    print (n);
    return 0;
}
```

```
void print ( int n){
    // base condition
    if ( n==0)
        return;
    // recurrence relation
    print (n-1);
    // processing
    cout << n;
}
```

## Recursion Tree




## Call Stack




Output - 1 2 3 4

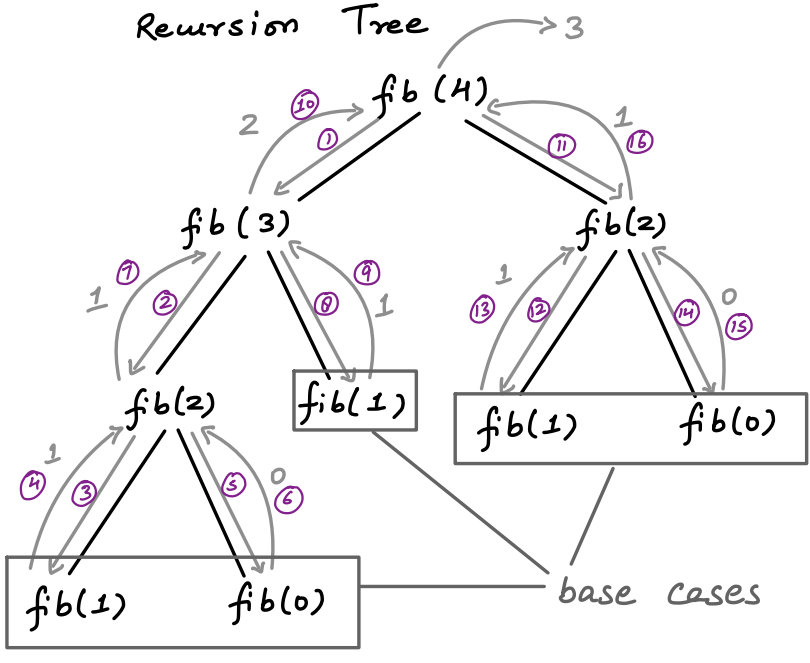
```
int main(){
    int n = ,
    int ans = fib(n);
    cout << ans;
    return 0;
}
```

```
int fib( int n){
    if ( n==0 || n==1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

1 case solve krdo

baaki recursion sambhal lega

| n ⟶ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| fib(n) ⟶ | 0 | 1 | 1 | 2 | 3 |




## Magical line for Recursion

Ek case solve krdo, baaki recursion sambhal lega

- Just believe on it, dont doubt on recursion
  - Dont go depth inside recursion tree

# W7-L2

→ func ( vector <int > arr , int i , int & ans)

pass by reference

if we want to retain value of ans after function call

→ Try to pass everything pass by reference IF POSSIBE

↳ SC ↓es

TC ↓es (no copying of variables again & again)

→ to reverse answer, you can use recursion

**Integer literal with a leading 0**

n= 0100 ; ⟶ interpretted as an octal number

cout << n; ⟶ 64 ⟶ convert 0100 octal to decimal

cin >> n ; ⟶ ( 0 1 00)

cout << n; ⟶ 100 ⟶ no conversion as cin reads 0100 as 100